Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.050425

ARTICLE

Preserving Data Secrecy and Integrity for Cloud Storage Using Smart Contracts and Cryptographic Primitives

Maher Alharby*

Cybersecurity Department, College of Computer Science and Engineering, Taibah University, Medina, 42353, Saudi Arabia

*Corresponding Author: Maher Alharby. Email: mharby@taibahu.edu.sa

Received: 06 February 2024 Accepted: 29 March 2024 Published: 15 May 2024

ABSTRACT

Cloud computing has emerged as a viable alternative to traditional computing infrastructures, offering various benefits. However, the adoption of cloud storage poses significant risks to data secrecy and integrity. This article presents an effective mechanism to preserve the secrecy and integrity of data stored on the public cloud by leveraging blockchain technology, smart contracts, and cryptographic primitives. The proposed approach utilizes a Solidity-based smart contract as an auditor for maintaining and verifying the integrity of outsourced data. To preserve data secrecy, symmetric encryption systems are employed to encrypt user data before outsourcing it. An extensive performance analysis is conducted to illustrate the efficiency of the proposed mechanism. Additionally, a rigorous assessment is conducted to ensure that the developed smart contract is free from vulnerabilities and to measure its associated running costs. The security analysis of the proposed system confirms that our approach can securely maintain the confidentiality and integrity of cloud storage, even in the presence of malicious entities. The proposed mechanism contributes to enhancing data security in cloud computing environments and can be used as a foundation for developing more secure cloud storage systems.

KEYWORDS

Cloud storage; data secrecy; data integrity; smart contracts; cryptography

1 Introduction

Cloud computing has arisen as an alternative to traditional computing infrastructures. It offers users the convenience of accessing a diverse range of computing resources, such as storage, servers, and applications, as per their requirements via the Internet. This computing paradigm encompasses three core service models: Platform as a Service, Infrastructure as a Service, and Software as a Service. Among the various offerings, cloud storage services present unique advantages. These include resource accessibility from any location, cost-effectiveness, and the removal of the hassle associated with managing hardware and software in-house. However, the transition to cloud storage poses significant risks to data security, specifically data secrecy and integrity. A recent report has shown that 73% of companies are concerned about cloud security [1]. Data secrecy refers to data protection from unauthorized access, which is typically achieved through cryptographic methods. On the other hand,

data integrity ensures that data remains unaltered and authentic, which is often maintained using hash functions and digital signatures.

Verifying the integrity of data stored on the cloud poses a significant challenge for users as they usually delete their local copies once data has been uploaded to the cloud [2]. Regularly downloading data for integrity checks is not practical due to bandwidth constraints [3]. That is, many researchers have proposed the use of a third-party auditor to check the integrity of cloud storage. Ensuring the security of third-party auditors is an essential challenge due to the potential for malicious behavior. Recently, researchers have started exploring blockchain technologies to eliminate the reliance on third-party auditors. Blockchain technology provides a secure, transparent, and trustless method for verifying transactions in various applications, including supply chain management [4], healthcare [5], and cloud storage [6]. Smart contracts can be utilized on top of the blockchain network to replace traditional third-party auditors, ensuring data integrity and promoting honesty. However, existing research studies suffer from inefficiency as they require extensive computation overhead and do not adequately address the privacy of user data files.

In this article, we aim to address the limitations of current research by introducing an efficient and privacy-preserving mechanism that guarantees both the integrity and secrecy of data stored on the public cloud. To achieve this, we leverage blockchain and smart contract technologies to perform periodic data audits, as they are well-known for their ability to provide data immutability. In addition, we leverage cryptographic primitives to maintain data secrecy by encrypting the user data stored on the cloud.

The contributions of this article are as follows. Firstly, we propose an efficient scheme to preserve the confidentiality and integrity of cloud storage by leveraging smart contracts, blockchain oracles, and symmetric cryptographic algorithms. Smart contracts and blockchain oracles are primarily used to verify data integrity, while cryptographic algorithms are used to maintain the secrecy of outsourced data. Secondly, we implement the proposed system, including its smart contract and oracle interface. We implement the system's smart contract as a data auditor using the Solidity language. We employ a Chainlink oracle interface to facilitate communication between the smart contract and the cloud storage. Thirdly, we conduct extensive performance analysis to demonstrate the efficiency of our system. Fourthly, we conduct a vulnerability assessment to ensure that the developed smart contract is free from bugs and errors and measure the cost associated with running such smart contracts. Additionally, we discuss how our proposed scheme can securely preserve the confidentiality and integrity of cloud storage, even in the presence of malicious entities.

This article is structured as follows. Related work is discussed in Section 2. We introduce the design and implementation of our proposed scheme in Sections 3 and 4. In Section 5, we present the findings concerning performance evaluation and smart contacts. We provide a security analysis of the system in Section 6 and conclude the article in Section 7.

2 Related Work

2.1 Data Integrity Verification Using Third-Party Auditors

Previous studies led by Juels et al. [7] have developed the concept of Proofs of Retrievability, which allows a verifier to confirm the authenticity and retrievability of a specific file. Shacham et al. [8] have built on this concept by integrating Boneh-Lynn-Shacham (BLS) signatures and pseudorandom functions, resulting in reduced times for queries and responses. Additionally, Ateniese et al. [9] have introduced the Provable Data Possession model, which allows users to verify if their data, hosted

on untrustworthy servers, is unaltered without retrieving it. These approaches, however, require continuous communication between the user and the service provider, which can lead to significant performance burdens for the user.

To mitigate the impact on user performance, incorporating a third-party auditor for verification purposes has been suggested. For instance, Zhu et al. [10] introduced a data integrity verification approach that uses a short signature algorithm, ensuring privacy while enabling public auditing via a centralized auditor. Similarly, Ping et al. [11] developed a scheme for data integrity checks using algebraic signatures and elliptic curve cryptography. This approach delegates the role of verifying outsourced data integrity to an intermediary. Moreover, Lu et al. [12] proposed a more efficient auditing method where the third-party auditor generates block tags and conducts integrity checks, relieving the user of these tasks. Furthermore, Huang et al. [13] designed a privacy-preserving data integrity system that takes different data auditing frequencies into consideration. The system aims to reduce the auditing costs that third-party auditors charge for users who do not regularly verify their data files. However, ensuring the security of third-party auditors remains a critical issue that this article aims to address.

2.2 Data Integrity Verification Using Blockchain Technology

In the realm of blockchain-based research, a significant focus has been on applications within cloud storage that replace traditional third-party auditors with blockchain technology to manage and permanently record auditing tasks, thus ensuring mutual honesty among participants. In this context, Huang et al. [14] introduced a collaborative blockchain system to audit cloud data storage. Here, instead of a single auditor, consensus nodes collectively perform auditing tasks and maintain data integrity against diverse threats. However, some models exclude blockchain from essential verification processes. Yue et al. [15] developed a blockchain scheme that utilizes Merkle trees and random challenge numbers to verify the integrity of cloud data. Wei et al. [16] created a system that allows multiple parties to collaboratively verify data trustworthiness by utilizing blockchain's smart contract features to track data modifications. Shu et al. [17] developed a decentralized framework that operates similarly to a traditional third-party auditor, but with reduced risks of compromised auditors and hostile blockchain miners due to its self-governing structure. Tian et al. [18] introduced a blockchain-enabled secure de-duplication and collective auditing methodology for distributed storage systems. This methodology incorporates a dual direction shared auditing process, facilitating decentralized public auditing without the need for a traditional intermediary. Zhao et al. [19] presented a blockchain-supported public auditing scheme, offering conditional anonymity and privacy preservation. It is designed to withstand man-in-the-middle attacks and ensure the integrity of stored data, along with protecting data privacy and offering conditional anonymity for users' identities. Guo et al. [20] devised a reversible, blockchain-aided attribute-based encryption mechanism with an escrow-free setup, addressing the key escrow issue by using federated blockchains instead of conventional key management entities. Kumari et al. [21] proposed a cloud-based data auditing method to assure the integrity of healthcare data. Furthermore, Li et al. [22] devloped a novel certificateless data ownership verification system, which emphasizes both effectiveness and privacy preservation. While the existing approaches have potential benefits, they suffer from two major limitations. Similarly, Li [23] introduced a blockchain-based privacy-preserving system to enhance logistics in the context of the Internet of Things. The proposed approach makes use of hash functions to detect data integrity, ciphertext-policy attribute-based encryption to achieve secrecy, and smart contracts to manage data access. Firstly, they are not very efficient as they require extensive computation and communication among the network participants. Secondly, the privacy of user data files is not properly addressed, which raises security

concerns. Our proposed scheme aims to overcome these issues and provide a more efficient and secure solution that guarantees the integrity and secrecy of data stored on the public cloud.

3 Proposed Data Secrecy and Integrity Scheme

In this section, we introduce the proposed mechanism for preserving data secrecy and integrity for cloud storage, emphasizing the main components of the system and the interaction among them.

3.1 High-Level System Model

This section describes the proposed system, at a high level, for preserving the secrecy and integrity of data files stored on the public cloud. The proposed system comprises five entities: Data owner, cloud storage, blockchain network, smart contract, and oracle interface. We explain the role of each entity and its interactions with other entities as follows:

Data owner (DO). This entity can be represented as an individual user or organization that wants to outsource their data files to the public cloud storage and also check the integrity of such outsourced files. To maintain data confidentiality, the data owner enciphers the data files before outsourcing them.

Public cloud storage (PCS). This entity is responsible for serving the data owner by storing and maintaining their data files. Since data stored on the cloud is controlled by a non-trusting third party, the data owner encrypts their data files before outsourcing them. In addition to providing storage services, it should also provide proof of data correctness whenever the data owner asks for it.

Blockchain network (BN). This entity represents the underlying peer-to-peer network. It enables the deployment and execution of smart contracts on top of it.

Smart contract (SC). This entity is represented as a script running on the blockchain network. It has a specific address to be called at. The data owner sends the metadata for a data file to be stored on the contract state. When the data owner wants to verify the integrity of their data files stored on the public cloud, they can issue an audit request through the smart contract to the public cloud. Furthermore, the smart contract can perform some computations to check whether the auditing response received from the cloud storage is correct.

Interface oracle (IO). This entity is represented as an off-chain entity that enables communication between the blockchain smart contract and public cloud storage. Smart contracts cannot communicate with the outside world except with the help of blockchain oracles. That is, we rely on an interface oracle for forwarding the auditing request from the smart contract to the cloud as well as delivering the auditing response from the cloud to the contract. In other words, it can be thought of as an intermediary between the smart contract and cloud storage.

The proposed auditing scheme's high-level workflow is illustrated in Fig. 1, covering both data outsourcing and auditing processes as follows. Firstly, to outsource data, the data owner must encrypt the data file and compute its metadata. The encrypted data file is then transmitted to the public cloud for storage, while its corresponding metadata is recorded on the smart contract for future data auditing purposes. Secondly, to audit an outsourced data file, the data owner can submit an audit request through the smart contract to the public cloud. The smart contract utilizes an oracle interface to send the auditing request and receive the auditing response from the public cloud. Once the smart contract receives the auditing response, it verifies the accuracy of the provided response and notifies the data owner about the integrity of their outsourced data files.

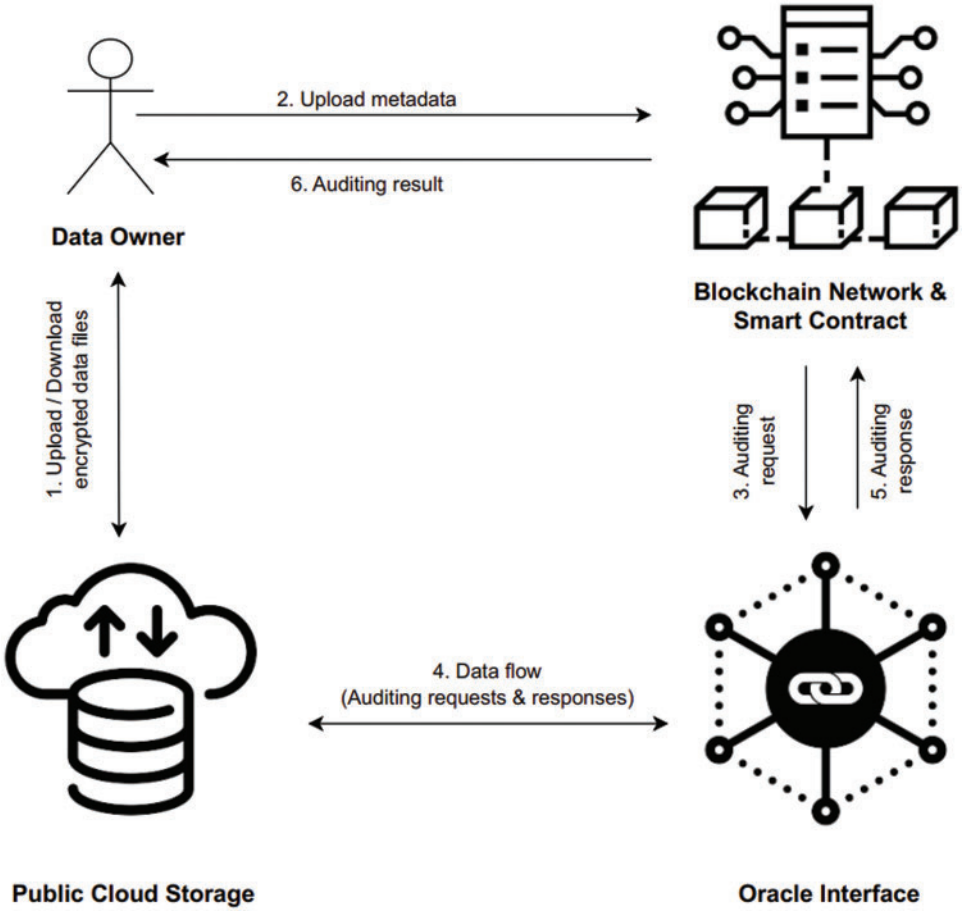

Figure 1: The proposed data secrecy and integrity scheme

3.2 System Requirements and Assumptions

We consider the following criteria as design requirements for our proposed scheme. First, the smart contract should be able to verify the integrity of outsourced data files without the need to store them on the blockchain or retrieve them from the public cloud. That means the smart contract does not need a copy of the data files to perform the auditing task. Secondly, the proposed scheme should preserve the secrecy of outscored data files by preventing everyone (except the data owner) from accessing such data files. Blockchain network nodes, for instance, should not be able to view or read the content of data files. In addition, the smart contract code should not have any bugs or vulnerabilities that can be utilized to access the content of data files or manipulate the auditing results.

Various assumptions have been considered in our proposed scheme, which are as follows. First, we treat the public cloud as a non-trusted entity. That is, we rely on a smart contract integrated with an oracle interface to audit the data files stored on the cloud periodically. Secondly, the smart contract is assumed to honestly perform the data auditing task. However, since the contract code is placed on a public blockchain network, one might exploit the contract code to view the content of data files. We assume the contract is securely designed to prevent such malicious activities. Finally and more

importantly, we assume that the oracle interface can securely deliver the communications between the smart contract and the public cloud.

3.3 The Proposed Scheme

This section describes the details of our proposed data auditing scheme. The first phase of our scheme concerns data preparation and storage, while the second phase concerns challenges the public cloud to provide proof of data correctness.

3.3.1 Data Storage Phase

In the data storage phase, the data owner encrypts the data before outsourcing it to the public cloud. We assume the data owner has access to a symmetric cryptographic algorithm such as the AES algorithm and has a secret key k_s for encryption purposes.

$$Enc(k_s, P) \rightarrow C \tag{1}$$

This data encryption algorithm *Enc* takes as inputs the secret key for the data owner k_s and the data P to be encrypted. It then computes the encrypted format of the data C. The data owner then calculates the metadata C_m for the encrypted data. This metadata will be used for data auditing purposes. Before generating the metadata for the encrypted data, the data owner can utilize Elliptic Curve Cryptography (ECC) to generate a pair of private and public keys.

$$ECC\,(a, b) \rightarrow N, n, G \tag{2}$$

This ECC algorithm takes two prime numbers a and b of size m as inputs (about 200 digits) and then computes the private key = (N) and the public key = $(n,\ G)$. N is calculated as $lcm\ (a + 1, b + 1)$, n is computed as $a \times b$, and G is calculated as a generator point of N. To compute the corresponding metadata C_m for the encrypted data, the data owner has to process the encrypted data C with the ECC parameters G and N as follows:

$$C_m = C \cdot G \,mod\, N \tag{3}$$

Once the data owner prepares C and C_m, it broadcasts C to the public cloud for storage and the C_m to the smart contract for future data auditing tasks.

3.3.2 Data Integrity Verification Phase

In the data integrity checking phase, the smart contract sends an audit request to the public cloud that maintains the data. Since the smart contract resides on the blockchain network, there is a need for an oracle interface to allow the contract to communicate with the outside world. That is, the smart contract is connected to an oracle interface, which can be utilized to send the auditing request to the cloud. The auditing request R_a can be computed as follows:

$$R_a = C_{ID}|k|G \cdot \mathrm{s} \tag{4}$$

where C_{ID} is the ID or the name of the encrypted data file, k is a temporary key, and $G{\cdot}s$ represents the multiplication of G with a random integer s. Once the public cloud receives R_a through the oracle interface, it must calculate the auditing response R_S as proof of data correctness. The cloud generates a random integer x with the help of a keyed random function f_k and then computes R_S as follows:

$$R_S = x \cdot C \cdot (G \cdot s)\ mod\, n \tag{5}$$

After generating the response, the public cloud sends the auditing response to the smart contract via the oracle interface. The smart contract then verifies the correctness of the generated auditing response as follows. It first generates a random integer x with the help of a keyed random function f_{k} and then computes V as follows:

$$V = x \cdot C_m \, mod \, n \tag{6}$$

Then, the smart contract computes R_{V} as follows:

$$R_V = s \cdot V \, mod \, n \tag{7}$$

If the calculated R_{V} matches R_{s}, then the integrity of the data is intact. The contract can then notify the data owner about the auditing result.

4 Implementation

In this section, we will delve into the specifics of our proposed scheme for maintaining the confidentiality and integrity of outsourced data. We will start by presenting the primary algorithms utilized for secure data storage and ensuring data integrity. Following that, we will examine the implementation of the smart contract.

4.1 Algorithms

The proposed scheme is executed in two distinct phases. The initial phase involves data preparation and storage and is primarily carried out by the data owner. On the other hand, the second phase is responsible for verifying data integrity.

Algorithm 1 shows the generic steps for implementing the first phase. Firstly, the data owner encrypts the data files and generates their verification metadata. We implement this phase using Python. For data file encryption, we leverage the Crypto.Cipher package by importing the AES class. We generate a key of 192 bit long k_{s} and then run the cipher. encrypt method to encrypt the data file. The encrypted data file C is then forwarded to the public cloud for storage. To generate the verification metadata, the data owner first selects two prime numbers and generates public and private ECC keys. To generate the ECC keys, we leverage the ECC class by importing the Crypto.PublicKey package. The verification metadata C_{m} is computed using the Python mod package. The data owner then sends C_{m} to the smart contract for any future data integrity checks.

Algorithm 1: Data preparation and storage on public cloud

Inputs: k_{s}, a, b, P
Outputs: C, C_{m}
procedure ENCRYPTING DATA FILES
    $k_{\mathrm{s}} \leftarrow$ Generate a symmetric key
    $C \leftarrow Enc(k_{\mathrm{s}}, P)$
    Send C to PC for storage
end procedure
procedure GENERATING ECC KEYS AND METADATA
    $a,b \leftarrow$ Select two prime numbers
    $N,n,G \leftarrow ECC(a,b)$
    $C_{\mathrm{m}} = C \cdot G \bmod N$

(Continued)

Algorithm 1 (continued)

12: Send C_m to SC for auditing purposes
13: end procedure

Algorithm 2 shows the generic steps for the implementation of the second phase. We implement this phase as a smart contract running on the Ethereum network and Python code running on the public cloud. We consider the Ethereum network over other private blockchains like Hyperledger since it is an open network where anyone can join and participate in the network. In addition, it offers high levels of transparency and security in comparison to private networks. To check the integrity of data files stored on the cloud, the data owner sends a transaction to the smart contract whenever the integrity of a data file needs to be checked. The transaction should contain the ID of the encrypted file, a temporary key, and the multiplication of a random integer and the ECC public key. The smart contract transaction is then executed by generating an auditing request R_a as a challenge and forwarding it to the public cloud through an oracle interface. We discuss the details of the smart contract implementation in the following section. Upon receiving the R_a, the public cloud storage computes the data auditing response. We implement the cloud as a server running a Python code to compute the auditing response. The cloud server generates a random integer using a keyed random function algorithm and then calculates the auditing response R_s by applying the modular arithmetic operation stated in Line 8. The cloud then sends R_s to the smart contract as proof of data correctness. When the smart contract receives R_s from the cloud, it executes a verification method to check whether the received response R_s is valid or not. The data owner will then be informed about the result of the data integrity check.

Algorithm 2: Data integrity verification

1: procedure AUDITING REQUEST
2: $R_a \leftarrow \text{CID}|k|G{\cdot}s$
3: DO sends R_a to PC via OI
4: procedure AUDITING RESPONSE
5: PCS receive R_a from OI
6: $x \leftarrow$ Generate random integer using f_k
7: $R_S \leftarrow x{\cdot}C{\cdot}(G{\cdot}s) \bmod n$
8: PCS sends R_s back to SC via OI
9: end procedure
10: procedure VERIFYING RESPONSE
11: CS receives R_s from OI
12: $x \leftarrow$ Generate random integer using the same f_k
13: $V \leftarrow x{\cdot}C_m \bmod n$
14: $R_V \leftarrow s{\cdot}V \bmod n$
15: if $R_V == R_S$ then
16: Data integrity is intact
17: Else
18: Data integrity is compromised
19: end if
20: Notify DO about integrity results
21: end procedure

4.2 Smart Contract Implementation

Our proposed auditing system heavily depends on smart contracts, which serve as the system's foundation. To implement the data auditing activities in the system, we utilize the Solidity language to create an Ethereum smart contract. This smart contract is responsible for managing the auditing requests and verifying the correctness of the auditing responses. The smart contract is designed to ensure complete transparency and immutability throughout the auditing process. We use the Remix tool [24] to develop and test the smart contract, which provides various features such as code editing, error debugging, and smart contract deployment. We leverage the Ethereum Sepolia test network [25] for deploying and executing the smart contract. This approach offers several advantages, including testing the smart contract using unreal tokens, which can help identify vulnerabilities before posting the smart contract on the main network. In this section, we discuss the development of the smart contract integrated into the auditing system.

The smart contract is responsible for issuing an auditing request, sending the request to the public cloud, obtaining the auditing response from the cloud, and verifying the received response. We present the detailed implementation of the two most important functions of our smart contract, which are issueAuditReq and verifyAuditRes. These functions are meant to highlight the key functionality and operation of the auditing system.

issueAuditReq Function. This function is triggered whenever a data owner sends a transaction to execute it. It accepts three input parameters (file ID, a key, and an integer value), concatenates them, and then produces the auditing request to be forwarded to the public cloud. We note that the three parameters are of different types. The file ID and the integer value are of type unit, whereas the key is of a byte type. In Solidity, there is no straightforward way to concatenate values of different types. That is, we leverage a Solidity function called abi.encodePacked to convert the integer values into bytes. Then, we use the bytes.concat method to concatenate the three byte sequences into a single-byte sequence representing the auditing request.

verifyAuditRes Function. This function is executed upon receiving the auditing response from the cloud to verify the response's correctness. To verify the auditing response, there is a need to generate a random integer x. We utilize the Keccak256 hash function to implement a keyed random function to generate x. Since the Solidity language does not directly support randomness, we use block properties such as the block's timestamp.

We leverage the Chainlink oracles [26] to enable communication between our smart contract and the public cloud. We import a Chainlink oracle interface called AggregatorV3Interface to pass the auditing request and retrieve the auditing response from the cloud through the Ethereum Virtual Machine.

5 Performance and Smart Contract Results

In this section, we present the performance and smart contract results of our proposed scheme. For performance results, we measure the computation complexity required by the data owner, smart contract, and public cloud. We conduct our performance experiments using a laptop with a Core i7 GHz CPU and 8 GB of RAM. Concerning smart contract results, we analyze and present the security results using the Slither vulnerability detection tool as well as present the associated transaction costs.

5.1 Performance Results

In this section, we calculate the computation costs, in seconds, required by the data owner, smart contract, and public cloud for different sizes of data files. Generally, the computational complexity of our proposed scheme is acceptable. The utilization of an oracle interface to connect both the smart contract and the public cloud adds an extra layer of computational complexity. Still, there is no other way to enable a smart contract to interact with an external entity except with blockchain oracles.

For the data owner, we measure the time it takes to encrypt the data file and generate its auditing metadata. We use the AES algorithm with a 192-bit key for data encryption. For metadata, a single modular multiplication operation is required. Fig. 2a shows the computation time in seconds for encrypting a data file and generating its metadata, considering different sizes (in Megabytes) of data files.

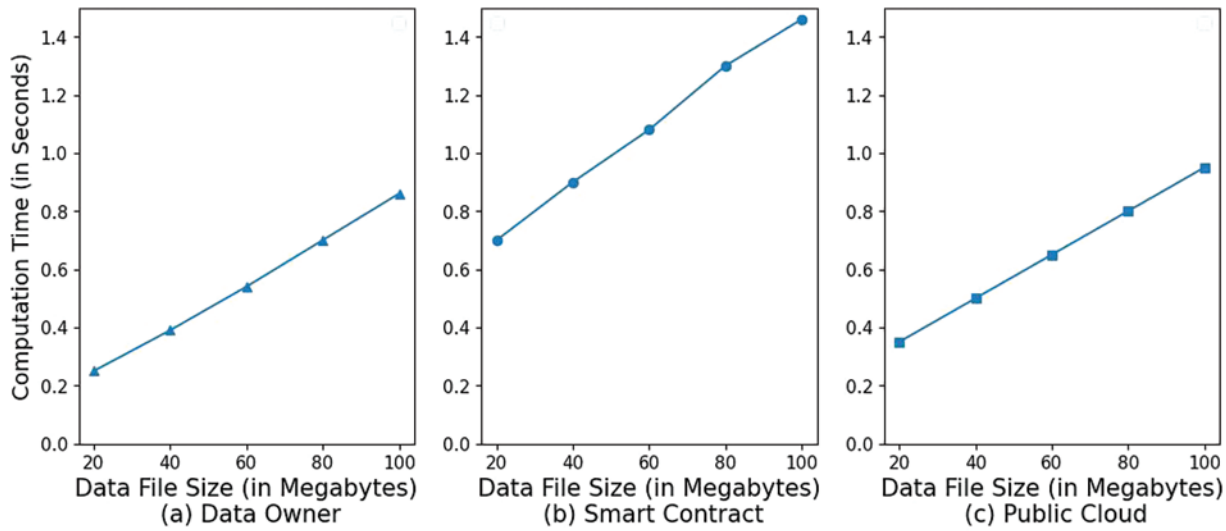

Figure 2: Computation time (in seconds) required by (a) data owner, (b) smart contract, and (c) public cloud for different data file sizes

For the smart contract, we measure the time it takes to issue an auditing request and verify the auditing response. To issue an auditing request, there is a need to generate a temporary key k and a random integer s in addition to performing a multiplication operation. After generating the request, the smart contract sends it to the cloud through an oracle interface. Once the auditing response is received from the cloud, the contract verifies the response by generating a random integer and performing two modular multiplication operations. Fig. 2b shows the computation time in seconds for issuing an auditing request and verifying the correctness of the received auditing response, considering the data file sizes (in Megabytes). We note that the computation time reported considers the network delay required to pass the auditing request and response from and to the oracle interface. The actual computation would be much less than that if we ignored the network delay.

For the public cloud, we measure the time it takes to compute the auditing response. The cloud needs to first generate a random integer in addition to performing a multiplication operation and a modular multiplication operation. Fig. 2c shows the computation time in seconds for computing the auditing response, considering the data file sizes (in Megabytes).

5.2 Smart Contract Results

This section presents the results of the smart contract that we have developed. We first show and discuss the vulnerability assessment results obtained from the Slither security auditing tool. Furthermore, we present the cost associated with deploying and executing the smart contract.

5.2.1 Vulnerability Results

This section analyzes the security of our smart contract to identify potential vulnerabilities and software bugs. We follow the Solidity security patterns during the development process of our smart contract. We also conduct extensive unit testing to ensure the correct functionality of the proposed scheme. In addition, we leverage a security tool named Slither [27] to look for potential software vulnerabilities. Slither performs a vulnerability scan for smart contracts and is compatible with most Solidity compilers. It can generate a vulnerability check report for a specific smart contract within a few seconds, making it convenient for developers to use. Table 1 shows the vulnerability assessment report for our proposed smart contract, produced by the Slither tool. Based on the generated report, we conclude that our smart contract code contains no software bugs or vulnerabilities.

Table 1: A summary of Slither vulnerability assessment results

Smart contract information		Vulnerability and informational issues	
Name	IntegrityVerification	Number of low issues	0
Number of functions	8	Number of medium issues	0
Source lines of code	80	Number of high issues	0
		Number of informational issues	2

5.2.2 Transaction Cost Results

This section reports the cost of executing smart contract transactions (as shown in Fig. 3). Since our proposed scheme is based on a smart contract running on the Ethereum blockchain [28], there is a cost associated with the auditing task. We first discuss the cost of smart contract deployment and then the cost of both the issueAuditReq and verifyAuditRes functions within the contract.

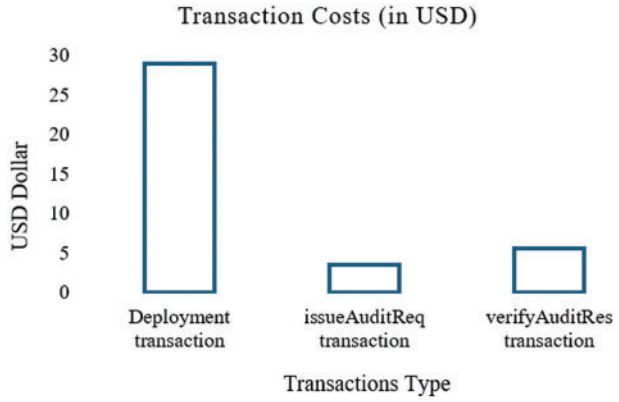

Figure 3: The cost (in USD dollars) for smart contract deployment, issueAuditReq and verifyAuditRes transactions

Smart contract deployment costs are paid once a transaction containing the smart contract bytecode is sent to the blockchain network. We measured the Gas Used cost for the deployment transaction and found it to be 1,034,716 units of Gas. The Gas units can be converted into the Ethereum currency by multiplying them by a Gas Price value, which the transaction submitter can set. The Gas Price value affects how fast transactions are confirmed by the network nodes. Let the Gas Price value is set at $1.2 \cdot 10^{-8}$ Ether, then the cost of the deployment transaction would be as follows: $1.2 \cdot 10^{-8} \cdot 1{,}034{,}716 = 0.0124$ Ether = \$28.96.

The cost of issueAuditReq and verifyAuditRes transactions is calculated as follows. We initiated a transaction to issue and submit an audit request by invoking the issueAuditReq function and found its Gas Used cost to be 127,653 Gas units. That is, the cost of the issueAuditReq transaction is as follows: $1.2{\cdot}10^{-8} \cdot 127{,}653 = 0.0012$ Ether = \$3.5. For the verifyAuditRes transactions, we initiated a transaction to confirm the correctness of the provided auditing response and found its Gas Used cost to be 202,024 Gas units. That is, the cost of the verifyAuditRes transaction is as follows: $1.2 \cdot 10^{-8} \cdot 202{,}024 = 0.0026$ Ether = \$5.6.

All transaction costs mentioned in this section are estimated using the Sepolia test network and the current implementation version of the proposed smart contract. The cost may differ depending on the current market price of Ether, the Gas Price amount chosen, and the level of optimization in the smart contract source code.

6 Discussion

In this section, we discuss the security of our proposed system with respect to data secrecy and integrity. This is to illustrate that the scheme can be utilized to audit outsourced data without leaking such data to unauthorized entities.

6.1 Data Secrecy

It is crucial to ensure the secrecy of user data, which is stored with a non-trusting entity called Public Cloud Storage (PCS). The primary concern is to prevent unauthorized access to the user's data by malicious PCSs or attackers while it is stored on the cloud. Additionally, it is important to protect the content of the user's data during the auditing process at the smart contract. In this section, we discuss how our designed scheme prevents data exposure to unauthorized parties.

Our scheme presents a formidable barrier to malicious parties attempting to access the user's data. The file is encrypted by the user using symmetric encryption primitives before being uploaded to the cloud, as explained in Section 3.3. Therefore, these malicious parties need knowledge of the secret key, which is securely held by the data owner, to decipher the file and access its contents. As a result, malicious parties are effectively prevented from accessing the user's data.

Hypothesis. The proposed mechanism prohibits unauthorized malicious parties from accessing the users' data files stored on the public cloud.

Proof. If a malicious party tries to gain access to the user's data files stored on the cloud, they will need to have the user's secret key. This is because all data files in our proposed system are enciphered with the user's secret key before being uploaded to the cloud. Whenever the secret key is protected from malicious parties, it is not feasible to gain any information about the user's data.

If a malicious party tries to access the data content from the metadata stored on the smart contract storage, it will not succeed because the verification metadata is computed using the owner's secret key. Malicious actors, therefore, must obtain the owner's secret key in order to access the user's data from

the metadata. Moreover, during the auditing process, the smart contract cannot gain knowledge of the data content from the received audit response since the response is not generated from the original data.

Our proposed scheme is highly effective in guarding against data leakage to malicious parties or potential attackers. This robust protection is primarily due to the encryption of the data file before its upload to the cloud.

6.2 Data Integrity

The proposed scheme permits data owners to verify the integrity of their outsourced data through reliance on smart contracts and blockchain oracles. This section illustrates to what extent the proposed scheme can accurately ascertain data integrity, assuming the honest behavior of PCSs. Subsequently, we will demonstrate the system's resilience against dishonest PCSs.

Hypothesis. Given that the PCS honestly computes the required auditing response for the auditing request received from the smart contract, the smart contract can deem the auditing response valid. We can show that R_V is equivalent to R_S by utilizing the commutative property of an elliptic curve as follows:

$$
\begin{aligned}
R_{V;} &= s \cdot V \bmod n \\
R_V &= s \cdot x \cdot C_m \bmod n \\
R_V &= s \cdot x \cdot C \cdot G \bmod n \\
R_V &= R_S
\end{aligned} \tag{8}
$$

Hypothesis. The PCS is not able to deceive the proposed system by incorrectly computing the auditing response.

Proof. The PCS may engage in dishonest activity, attempting to deceive the smart contract during the data integrity verification process. Here, we show that it is not feasible to deceive the proposed system by computing an invalid auditing response by discussing two potential malicious behaviors. The first possible malicious behavior is that the PCS may compute R_S without retaining the actual data. However, this is not feasible within our proposed scheme because generating the proof necessitates accessing the encrypted data file to produce a valid response during auditing. The second malicious behavior is that the PCS might consider reusing previous auditing responses as responses to the auditing requests received from the smart contract. Nevertheless, this behavior is impractical because the PCS would need to generate a fresh random integer for every response generated, as discussed in Section 3.3. That is, the PCS cannot use previous responses as they will be rejected by the smart contract as invalid.

In conclusion, it is evident from this discussion that our proposed scheme only accepts auditing responses when the PCSs honestly compute them. Furthermore, our scheme is resilient against any deceptive attempts by the PCSs during data auditing.

7 Conclusion

This article proposes an efficient mechanism to address the challenges of data confidentiality and integrity in cloud storage systems. We leverage blockchain and smart contract technologies to

periodically audit outsourced data. Furthermore, we employ symmetric encryption algorithms to ensure data confidentiality.

The contributions of this article are manifold. Firstly, we introduce a robust and efficient scheme that leverages smart contracts, blockchain oracles, and cryptographic algorithms to preserve the confidentiality and integrity of cloud storage. Specifically, smart contracts and blockchain oracles are utilized to verify data integrity, while cryptographic algorithms are used to maintain the secrecy of outsourced data. Secondly, we implement the proposed system, including its smart contract and oracle interface. We implement the system's smart contract using the Solidity language as a data auditor and a Chainlink oracle interface to enable communication between the smart contract and the cloud storage. Thirdly, we conduct extensive performance analysis to showcase the efficiency of our system. Additionally, we conduct a vulnerability assessment to ensure that the developed smart contract maintains the confidentiality and integrity of cloud storage, even in the presence of malicious entities. The scheme offers a strong foundation to enhance data security in cloud computing environments, and its implementation is expected to contribute to the development of more secure cloud storage systems.

Acknowledgement: The authors express their gratitude towards Taibah University for providing supervisory assistance.

Funding Statement: The author received no specific funding for this study.

Author Contributions: M. Alharby designed the proposed scheme, conducted experiments, analyzed results, and wrote the manuscript.

Availability of Data and Materials: All results are presented and discussed in the manuscript.

Conflicts of Interest: The authors declare that he have no conflicts of interest to report regarding the present study.

References

[1] H. Schulze, "Cloud security report," Cybersecurity Insider, 2021. Accessed: Feb. 29, 2024. [Online]. Available: https://www.cybersecurity-insiders.com/portfolio/2021-cloud-security-report-fortinet/

[2] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Trans. Comput.*, vol. 65, pp. 2386–2396, 2015. doi: 10.1109/TC.2015.2389960.

[3] F. Zafar *et al.*, "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends," *Comput Security*, vol. 65, pp. 29–49, 2017. doi: 10.1016/j.cose.2016.10.006.

[4] P. Dutta, T. M. Choi, S. Somani, and R. Butala, "Blockchain technology in supply chain operations: Applications, challenges and research opportunities," *Transport. Res. Part E Logist. Transport. Rev.*, vol. 142, pp. 102067, 2020. doi: 10.1016/j.tre.2020.102067.

[5] A. Saha, R. Amin, S. Kunal, S. Vollala, and S. K. Dwivedi, "Review on "Blockchain technology based medical healthcare system with privacy issues", *Security Privacy*, vol. 2, pp. e83, 2019. doi: 10.1002/spy2.83.

[6] P. Sharma, R. Jindal, and M. D. Borah, "Blockchain technology for cloud storage: A systematic literature review," *ACM Comput. Surveys*, vol. 53, pp. 1–32, 2020.

[7] A. Juels and B. S. Kaliski, "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. on Comput. Commun. Security*, Virginia, USA, 2007, pp. 584–597.

[8] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, pp. 442–483, 2013. doi: 10.1007/s00145-012-9129-2.

[9] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Security*, Virginia, USA, 2007, pp. 598–609.

[10] H. Zhu *et al.*, "A secure and efficient data integrity verification scheme for cloud-IoT based on short signature," *IEEE Access*, vol. 7, pp. 90036–90044, 2019. doi: 10.1109/ACCESS.2019.2924486.
[11] Y. Ping, Y. Zhan, K. Lu, and B. Wang, "Public data integrity verification scheme for secure cloud storage," *Inf.*, vol. 11, pp. 409, 2020. doi: 10.3390/info11090409.
[12] X. Lu, Z. Pan, and H. Xian, "An integrity verification scheme of cloud storage for internet-of-things mobile terminal devices," *Comput. Security*, vol. 92, pp. 101686, 2020. doi: 10.1016/j.cose.2019.101686.
[13] Y. Huang, W. Shen, J. Qin, and H. Hou, "Privacy-preserving certificateless public auditing supporting different auditing frequencies," *Comput. Security*, vol. 128, pp. 103181, 2023. doi: 10.1016/j.cose.2023.103181.
[14] P. Huang, K. Fan, H. Yang, K. Zhang, H. Li and Y. Yang, "A collaborative auditing blockchain for trustworthy data integrity in cloud storage system," *IEEE Access*, vol. 8, pp. 94780–94794, 2020. doi: 10.1109/ACCESS.2020.2993606.
[15] D. Yue, R. Li, Y. Zhang, W. Tian, and Y. Huang, "Blockchain-based verification framework for data integrity in edge-cloud storage," *J. Parallel Distr. Comput.*, vol. 146, pp. 1–14, 2020. doi: 10.1016/j.jpdc.2020.06.007.
[16] P. Wei, D. Wang, Y. Zhao, S. K. S. Tyagi, and N. Kumar, "Blockchain data-based cloud data integrity protection mechanism," *Future Gener. Comput. Syst.*, vol. 102, pp. 902–911, 2020. doi: 10.1016/j.future.2019.09.028.
[17] J. Shu, X. Zou, X. Jia, W. Zhang, and R. Xie, "Blockchain-based decentralized public auditing for cloud storage," *IEEE Trans. Cloud Comput.*, vol. 10, pp. 2366–2380, 2021. doi: 10.1109/TCC.2021.3051622.
[18] G. Tian *et al.*, "Blockchain-based secure deduplication and shared auditing in decentralized storage," *IEEE Trans. Dependable Secure Comput.*, vol. 19, pp. 3941–3954, 2021.
[19] J. Zhao, H. Huang, C. Gu, Z. Hua, and X. Zhang, "Blockchain-assisted conditional anonymity privacy-preserving public auditing scheme with reward mechanism," *IEEE Syst. J.*, vol. 16, pp. 4477–4488, 2021. doi: 10.1109/JSYST.2021.3125835.
[20] Y. Guo, Z. Lu, H. Ge, and J. Li, "Revocable blockchain-aided attribute-based encryption with escrow-free in cloud storage," *IEEE Trans. Comput.*, vol. 72, no. 7, pp. 1–12, 2023. doi: 10.1109/TC.2023.3234210.
[21] D. Kumari, P. Kumar, and S. Prajapat, "A blockchain assisted public auditing scheme for cloud-based digital twin healthcare services," *Cluster Comput.*, 2023. doi: 10.1007/s10586-023-04101-y.
[22] R. Li, X. A. Wang, H. Yang, K. Niu, D. Tang and X. Yang, "Efficient certificateless public integrity auditing of cloud data with designated verifier for batch audit," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, pp. 8079–8089, 2022. doi: 10.1016/j.jksuci.2022.07.020.
[23] K. Li, "Privacy-preserving scheme with bidirectional option for blockchain-enhanced logistics internet of things," *IEEE Internet Things J.*, 2024. doi: 10.1109/JIOT.2024.3370375.
[24] "Ethereum," *REMIX Ethereum IDE*, 2023. Accessed: Dec. 12, 2023. [Online]. Available: https://remix.ethereum.org
[25] "Ethereum Foundation," *Sepolia Testnet*, 2023. Accessed: Dec. 27, 2023. [Online]. Available: https://github.com/eth-clients/sepolia
[26] L. Breidenbach *et al.*, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Chainlink Labs*, vol. 1, pp. 1–136, 2021.
[27] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd Int. Workshop on Emerg. Trends in Softw. Eng. for Blockchain (WETSEB)*, Montreal, QC, Canada, 2019, pp. 8–15.
[28] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.